2. Design algorithm - Knowing all the logics is the reason of studying algorithm. Which logic to apply comes under design. DP, GS or D&C which to apply
3. Flow-chart
4. Verification (Manually) ↓ algorithm till now
5. Implementation to Coding ↓ Get program.
6. Analysing algorithm (time and space complexity) [CPU time, memory space]

Chapter 1. : ANALYSIS OF ALGORITHMS

Why analysis - To solve a given problem, if more than one algorithms are there, then to choose the best among them, we need analysis. based on two factors 1. Time 2. Space
Time complexity is more important as CPU time is costly than main-memory space.

Time complexity -

$T(P) = C(P) + R(P)$ compile time + running time

C(P) → Compiler → S/W → PL dependent
R(P) → CPU → H/W → Type of processor dependent

Analysis

Apostiary (relative)	Apriori (absolute)
1. PL-compiler and Type of H/W dependent Postponing	1. PL-compiler and Type of H/W independent Priority
2. Exact	2. Approximate
3. Diffesunt - vary from comp. to comp.	3. Same time in every computer.
4 Not much used.	4. Widely followed by software industry
5 Credit to h/w or PL for less time / faster program (super-comp.)	5. Credit to the programmi who gave the logic. (super-algorithm)

≡ Apriori Analysis → (absolute)

It is a determination of order of magnitude of a statement. (no. of times a statemer is executed while running

Ex1.
```
main()
{ x=y+z; ⇒ 1
}
```
O(1)

Ex2
```
main()
{ x+y+z → 1st ——→ 1
for (i=1; i<=n; i++)
{
a=b+c; } → 2nd ——→ n.
}
```
n+1 = O(n)
[logic matters]

Ex3
```
main()
{ x=y+z; ——→ 1
for (i=1; i<=n; i++)
{ x=y+z } ——→ n
```

```
for (i=1; i<=n; i++) {
  for (j=1; j<=n; j++)
  { x=y+z; }
}
```

$\longrightarrow n^2$ (n·n)

$n^2+n+1 = O(n^2)$

Ex 4 $100+1 = 101 = O(1)$.

$n-1 = n \approx O(n)$ — Approximate.

Undefined time complexity - means no algorithm.

```
main()
{  i=1;
while (i<=n) {
    i=i+10;
  }
}
```

$n/10 = O(n)$

Time complexity concerns on the main part of program i.e. loops and the most largest loop. No loops, 1 constant complexity.

```
main() {
  while (n>=1)
  { n=n-10
    n=n-20
    n=n+15
  }}
```

$\}$ $n=n-15 \Rightarrow n/15 = O(n)$

Ex 5

```
main() { i=1
  while (i<n)
  {
    i=2*i;
  } }
```

$(i \le n)$

$\Rightarrow \log_2 n \Leftarrow \log_2 n+1$

$O(\log_2 n)$

$1, 2, 2^2, 2^3 \ldots 2^k$

$2^k = n$

$\log_2 2^k = \log_2 n \Rightarrow k = \log_2 n$

$1, 3, 3^2, 3^3, \ldots 3^k$

$\log_3 n$

$i = 2i \Rightarrow \log_2 n$

$i = 3i$

for $i = i/2$ same story $\log_2 n$

$n \quad n/2 \quad n/2^2 \quad n/2^3 \; -- \; n/2^K$

$\frac{n}{2^K} = 1 \Rightarrow n = 2^K ; \; K = \log_2 n$

In context of division, subtraction will not affect much. Division will affect more.

Ex.6
```
main() {  i=2
while( i<n)
  {
    i = i^2;        ---> log(logn)
  }
}
```

$2, (2^2)^1, (2^2)^2, (2^4)^2 \; --- \; (2^{2^K})$

$2^{2^K} = n$

$K = \log_2(\log_2 n)$

(initial value, exponent value)

Square root

```
while(n>=2)        <- initial
{
  n = n^1/2        --->  log(logn)
}                   <- exp.
```

$n, \; n^{1/2}, \; (n^{1/2})^{1/2}, \; (n)^{1/2^3} \; --- \; (n)^{1/2^K}$

$\frac{n}{2^K} = 2$

$\frac{1}{2^K} \log_2 n = 1 \Rightarrow K = \log_2 \log_2 n$

Power has the most affecting. So ignore the other mul., div, add or sub.

Page No. 6
Date : 13/02/16

Gate problems -

find time complexities →

1.
```
main()
{
for (i=1; i<=n^3; i++)
{
for (j=n; j>1; j=j/5)
{
for (k=10; k<=n^5; k=k^15)
{
x=y+z;
}}}
```

$n^3 \log_5 n \log_{15} \log_{10} n^5$

2.
```
main()
{  p=1, q=1
for (i=1; i<=n; i=i*2)
    p++
for (j=1; j<=p; j=j*10)
    q++
printf(q);
}
```

Time complexity - $\log_2 n$ (largest of two loops)

3.
```
main()
{
for (i=1; i<=n^3; i++)
{
for (j=1; j<=n^2; i++)
```

{
for (i=1; i<= n^4 ; i++) {
x=y+z;
}}}
Time complexity is $1 \cdot n^4 = O(n^4)$

4. main()
{
for (i=1; i<=n; i++) ⟶ n
{
for (j=1; j<=i; j=j*2) ⟶ $\log_2 n$
{
for (k=1; k<=100; k++) ⟶ 100
{ x=y+z;
}}}
i & j are dependent loops

i=1	i=2	.....	i=n
log(1).O(1)	log(2).O(1)	.....	$\log_2(n).O(1)$

Time complexity -
$O(1)[\log 1 + \log 2 + \cdots \log n]$
$O(1)[\log(1+2+\cdots n)]$
$= \log(n!)$.
If k<=n, then $T(P) = n \log(n!)$.

5. main()
{ int i;
for (i=1; i<=n; i++)
{
for (j=1; j<=i^2; j++)
{
for (k=10; k<=n^5; k=k^50) ⟶ $\log_{50} \log_{10} n^5$

Page No. 8
Date : 13/02/16

{ x = y+2
K = K+5
K = K*5
}}}

i and j are dependent & K is independent.

i = 1	i = 2	i = 3		i = n
j = 1^2	2^2	3^3		n^2

$TC = (1^2+2^2+\dots+n^2)\log_{50}\log_{10}n^5$

$= \frac{n(n+1)(2n+1)}{6}\log_{50}\log_{10}n^5$

$= n^3\log_{50}\log_{10}n^5$

5. main() {
for (i=1; $i^5 <= n^{10}$; i++)
{
for (j=1; $j^2 <= n$; j++)
{
for (K=1; $K^{50} <= n^{150}$; K++) ⟶
{ x = y+z;
}}}

i	1^5	2^5	3^5 ----	$(n^2)^5$
j	1^2	2^2	3^2 ----	$(n^{1/2})^2$
K	1^{50}	2^{50}	3^{50} ----	$(n^3)^{50}$

$TC = n^2 \cdot n^{1/2} \cdot n^3 = n^{11/2}$

```
A(n) {
if (n<=1) return(1)
else
return (A(n-1))
}
```

Recursion takes more spa
than non-recursion. (stac
Time will be same in
both cases.

How many times it get called? - TC.

O(n).

if -- return (A(n-10)) -- then TC = O(n/10) = O(n
return (A(n/2)) -- then TC = O(log₂n)
A(n/2+5) -- then TC = O(log₂n)
A(n^1/2) -- TC = log₂(log₂)n

Asymptotic Notations -

1. Big-oh-notation → (O)
2. Omega notation → (Ω)
3. Thetha notation → (θ).

Let f(n) && g(n) be two +ve functions

O-notation →

f(n) = O(g(n)) iff
f(n) <= cg(n) ∀n ; n≥n₀
such that ∃ 2 const. c > 0 & n₀ ≥ 1

Ex1

f(n) = n²+n+10 g(n) = n²
f(n) = O(g(n)) ⇒ f(n) <= cg(n)
n²+n+10 < c (n²)
⇓
3 n₀ = 3.

n²+n+10 = O(n²) c=3, n₀=3.

Ex-2 $f(n) = n^2$ $g(n) = n^2$

$f(n) = O(g(n))$

$n^2 \leq c\,n^2$ $\forall n,\ n \geq n_0$

$\Downarrow$ 1. $\Downarrow$ 1

$n^2 = O(n^2)$ $n_0 = 1,\ c = 1.$

Ex-3 $f(n) = n+10$ $g(n) = n-10$

$f(n) = O(g(n))$

$n+10 = c\,(n-10)$ $n_0 = 30.$

$\Downarrow$ 2

$n+10 = O(n-10)$ $c = 2,\ n_0 = 30$

Ex-4 $f(n) = n^2$ $g(n) = n$

$f(n) = O(g(n))$

$n^2 = c \cdot n$ $\forall n,\ n \geq n_0$

$\Downarrow$

n → not a constant

$n^2 \neq O(n)$

$100 = O(1)$

↳ greater bcoz of c value

$1 = O(100)$ → RHS greater

$100 = O(1/100)$ → greater bcoz of c value.

$n = O(n^2)$ $c = 1$, naturally greater.

Ω-notation :- $f(n) = \Omega\, g(n)$ iff

$f(n) >= c \cdot g(n)$ $\forall n,\ n \geq n_0$

such that $\exists$ 2 const. $c > 0$ $n_0 \geq 1$

Ex-1 $f(n) = n^2 + n + 10$ $g(n) = n^2$

$f(n) \geq c \cdot g(n)$

$n^2 + n + 10 \geq c\,n^2$

$\Downarrow$ 1.

$n^2 + n + 10 = \Omega(n^2)$ $c = 1,\ n_0 = 1$

Ex-2. $f(n) = n^2$ $g(n) = n^2+n+10$

$f(n) \geq c\, g(n)$

$n^2 \geq c_1 (n^2+n+10)$ c_1 may be fractional.

$\Downarrow 1/2$

$n_0 = 4$ $c_1 = 1/2$

$n^2 = \Omega(n^2+n+10)$

Ex-3 $f(n) = n$ $g(n) = n^2$

$f(n) \geq c\, g(n)$

$n \geq c \cdot n^2$

$\hookrightarrow 1/n$ not possible

$n \neq \Omega(n^2)$

Theta Notation →

$f(n) = \Theta(g(n))$ iff

1. $f(n) \leq c_1 g(n)$ for $n \geq n_0$
2. $f(n) \geq c_2 g(n)$ $n \geq n_0$

Ex 1 $f(n) = n^2+n+10$ $g(n) = n^2$

1. $n^2+n+10 \leq c_1 (n^2)$

$\Downarrow 2.3.$ $n_0 = 3$

2. $n^2 \leq c_2 (n^2+n+10)$

$\Downarrow 1$ $n_0 = 1$

$n_0 = 3.$

$n^2+n+10 = \Theta(n^2)$ $n_0 = 3$

Ex 2. $f(n) = n$ $g(n) = n.$

1 $n \leq c_1 (n)$

$\Downarrow 1$ $n_0 = 1$

2 $n \geq c_2 (n)$

$\Downarrow 1$ $n_0 = 1$

$n_0 = 1.$

$n = \Theta(n)$

Ex 3. $f(n) = 1/n$ $g(n) = n.$

1. $1/n \leq c_1 n$

$\Downarrow 1$

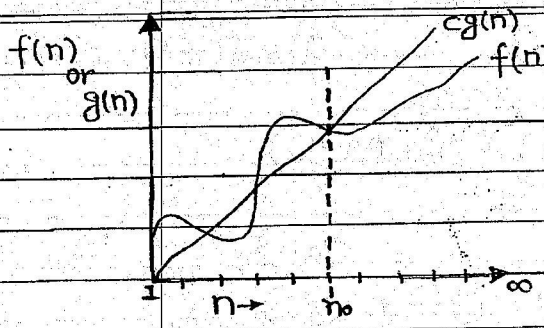
$n_0 = 1$ $\frac{1}{n} = O(n)$.

2. $1/n \geq c_2 n$

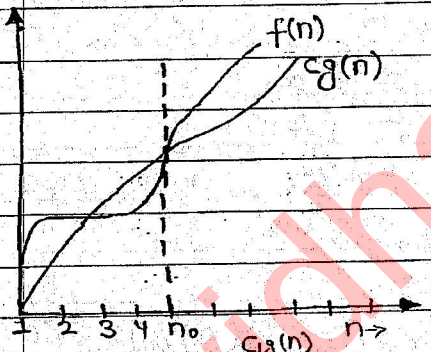
$\Downarrow 1/n^2$

Not possible

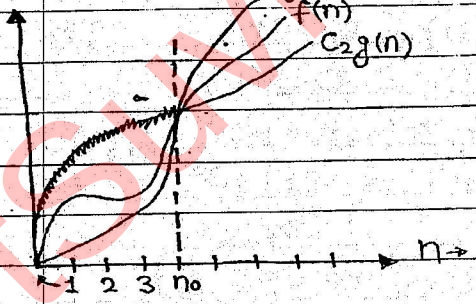
Big oh



Omega



Theta.



$n^2 = O(n^2)$ → Tightest upper bound (equal)
$= O(n^3)$ ⇒ upper bound (greater or equal to)
$= O(n^{10})$ → Not tightest upper bound (greater)

$A = O(B)$
⇓ UB → T
NT → small 'o'

Small-oh Notation (o) - operator - (<)
$A = o(B)$

Small-omega Notation (ω) - operator - (>)
$A = \omega(B)$

$n^3 = \Omega(n^2)$ Lower bound (smaller or equal)
$\Omega(n^3)$ Tightest lower bound (smaller)

Page No. 1
Date : 14/02/16

14-feb-16

Types of complexities -

Algorithm with time complexity constant is better on

1. constant $O(1)$ → loglogn
2. logarithmic $O(\log n)$ → n
3. Linear $O(n)$ → nlogn
4. Quadratic $O(n^2)$
5. Cubic $O(n^3)$
6. Polynomial) $O(n^c)$; c is a constant $c \geq 0$
7. Exponential $O(c^n)$; c is a constant $c > 1$.
8. $n! = O(n^n)$; $n! = o(n^n)$
   $[n^n > n!]$
9. $2^n < n^n$; $2^n = o(n^n)$; $n^n = \omega(2^n)$
10. $2^n < n! < n^n$

$[T(n) = \Theta(n)$ Asymptotically equal$]$
$n+10 = \Theta(n)$

Big-oh is the super-set of small oh

* - To prove small oh notation, show big-oh possible, omega not possible.

$2^n = O(3^n)$
$2^n \neq \Omega(3^n)$
So, $2^n = o(3^n)$

$2^n < 3^n$
$3^n \Rightarrow (2 \times 1.5)^n$
f^n factor $\quad 2^n \times (1.5)^n$

Greater by function means no Θ

11. $2^n = o(3^n)$; $2^n = O(3^n)$

$5^n < 7^n$
$7^n \Rightarrow \left(5 \times \frac{7}{5}\right)^n$
f^n factor $\quad 5^n \times \left(\frac{7}{5}\right)^n$

* $\log_2 n > \log_3 n$
[more base, less value]
$\Rightarrow \frac{\log_2 n}{\log_2 3}$
$\Rightarrow \frac{\log_2 n}{1.5}$ → Const. factor
$\log_2 n$

12. $\log_2 n = \Theta(\log_3 n)$
no small oh, small omeg

* When difference is a constant factor, then the functions are asymptotically equal (Θ).

$n > \log n$

$n > (\log n)^2$; $n > (\log n)^{100}$

[To check, apply log]

13. $n < (\log n)^{\log n}$.

$\sqrt{n} > \log n$ [Apply log both sides]

Gate problems -

1. Check True/false

a. $1000\, n\log n = O\left(\frac{n\log n}{100}\right)$ T

b. $\sqrt{n} = O(\log n)$ F

c. if $x<y$ then T

$n^x = O(n^y)$

d. $2^n \neq O(n^c)$ where c is const. T

2. Check True/False -

a. $(n+k)^m = O(n^m)$ where k & m are const. $T_{(m+1)}$

b. $2^{n+1} = O(2^n)$ T Ω, Θ

c. $2^{2n} = O(2^n)$ F Ω, ω

d. $f(n) = O((f(n))^2)$ T for increasing f^n

F for decreasing f^n

* Before applying log, first simplify/ cancel common terms.

$\frac{2^{2n}}{2^n} \Rightarrow 2^n$ $\frac{1}{1}$

→ for false, one enough

→ for true, all needed

3. Check True/False -

a. $n^3\, 32^{\log_2 n} = \Theta(n^8)$ T O, Ω, Θ

b. $64^{\log_8 n}\, n^3 = O(n^9)$ T O, Ω, Θ, o

c. $\frac{4^n}{2^n} = \Theta(2^n)$ T

d. If $f(n) = O(g(n))$
then $2^{f(n)} = O(2^{kg(n)})$ F o,w $\lceil b^{\log_c a} = a^{\log_c b}$
(Take $f(n) = 2n$ $g(n) = n$).
$\lceil n = O(n^2)$
$2^n = O(2^{n^2})$ ✓

4. $f(n) = \log(\log^* n)$
$g(n) = \log^*(\log n)$ Relⁿ betⁿ them?

$\log_2^*(1024)$ ✗ $\log_2 1024 \to 10$
$\Downarrow$ 4 $\log_2 10 \to 4 \Rightarrow \log_2 4$
$\Downarrow$
$1 \Leftarrow \log_2 2 \Leftarrow 2$

How many log needed to get the answer 1?

$\log_2^*(2^{2^{2^{2^2}}}) = 5$

$\log_2^* 2^{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}$ (11 lakh) = 1 lakh

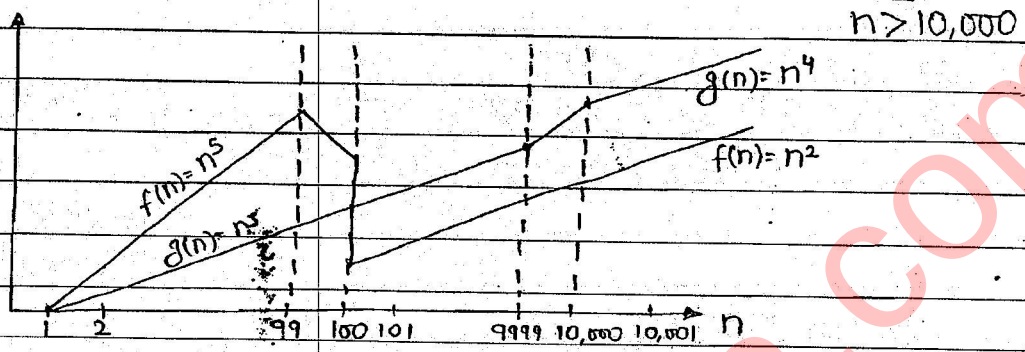
$f(n) = \log(\log^* n)$
$= \log(1,00,000)$
$= 15$

$g(n) = \log^*(\log n)$
$= \log^*(2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ (11 lakh)
$= 99,999$

Ans: $f(n) = O(g(n))$ o NO ω, Ω, Θ

5. Consider the following functions-
$f(n) = n^5$ $0 < n < 100$
$= n^2$ $n \geq 100$
$g(n) = n^3$ $0 < n < 10,000$
n^4 $n \geq 10,000$
Relⁿ b/w them?
$f(n) = \Omega(g(n))$ $0 < n < 100$
$f(n) =$ $100 < n < 10,000$
$10000 > n$.

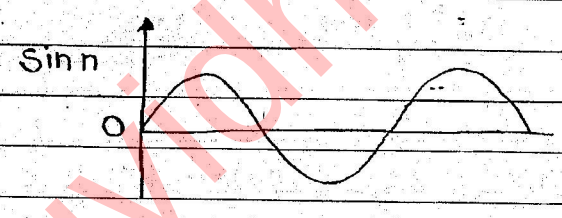
Ans: $f(n) = O(g(n))$ [final state]



6. $f(n) = n^{2+\sin n}$

$g(n) = n^{1+\cos n}$

n	f(n)	g(n)
90	90^3	90
180	180^2	1
270	270	270
360	360^2	360^2



$f(n) = \Omega(g(n))$ big-omega

$n_0 = 90$

n	f(n)	g(n)
90	90	1
180	1	1/180
270	1/270	1
360	1	360

7. $f(n) = n^{\sin n}$

$g(n) = n^{\cos n}$

Not comparable functions

So, no relations.

Properties of Asymptotic Notations →

1. Reflexive

$f(n) = O(f(n))$ O, Ω, Θ will satisfy

o, ω not satisfy.

2. Symmetric.

O, Ω will not satisfy

Θ satisfies.

3. Transitive $f(n) = O(g(n))$ $g(n) = O(h(n))$
then $f(n) = O(h(n))$
$O, \Omega, \Theta, o, \omega$ will satisfy.

4. If $f(n) = O(g(n))$
then $h(n)f(n) = O(h(n)g(n))$

5. If $f(n) = O(g(n))$ && $d(n) = O(e(n))$, then
(i) $f(n) + d(n) = \max(g(n), e(n))$
(ii) $f(n) \cdot d(n) = O(g(n) \cdot e(n))$
(iii) $f(n) / d(n) = O(g(n))$ both increasing f

Problem-1 Let $f(n)$, $g(n)$, && $h(n)$ be 3 +ve functions as:
(i) $f(n) = O(g(n))$ && $g(n) = O(f(n))$ [$f(n) = g(n) < h(n)$
(ii) $g(n) = O(h(n))$ && $h(n) \neq O(g(n))$ find relation
Consider True/false
(a) $f(n) = O(h(n))$ - T
(b) $f(n) \cdot h(n) \neq \Theta(h(n) \cdot g(n))$ T O, Ω
(c) $g(n) \cdot h(n) = \Theta(h(n) \cdot h(n))$ F O, o
(d) $h(n) = \Theta(f(n))$ F Ω, ω

2. Consider $T_1(n)$, $T_2(n)$ & $f(n)$ be three +ve functions
(i) $T_1(n) = O(f(n))$
(ii) $T_2(n) = O(f(n))$ then check the following.
(a) $T_1(n) = O(T_2(n))$ F
(b) $T_1(n) = \Omega(T_2(n))$ F [if one, then fals
(c) $T_1(n) = \Theta(T_2(n))$ F . for true, check al
(d) $T_1(n) + T_2(n) = O(f(n))$ T (property 4)